AMENDMENTS TO THE CLAIMS:

This listing of claims will replace all prior versions, and listings, of claims in the application:

LISTING OF CLAIMS:

1.(currently amended) A method for generating application software for managing a process, said method implementing a system software that is common to all of the application software generated by said method, said method comprising:

cybernetically representing said process in a set [[for]] of tasks and in relationships between the tasks;

breaking down each task into sub-blocks of tasks, until all actions of the process are described using at least one diagram composed of components including a root, branches, nodes and leaves, whereby each of the components is represented by an action corresponding to a generic object of an ~~attributing~~ attributed class;

transcribing each ~~object~~ component of each diagram into an attributed generic object by capturing data in predetermined formats associated with said attributes by using a capture interface associated to each class of generic object;

automatic precompiling to verify that the attributed objects required for an operation logic of the application are present and are supplied appropriately in terms of syntax;

automatic compiling during which data description of the attributed objects are integrated and are assembled with the system software to produce an executable application software; and

executing executable software of the application.

2.(previously presented) The method according to claim 1, wherein during the object transcribing stage, at least one action launches a complete processing cycle that is located at a remote location of a tree structure that corresponds to at least one diagram and, once said processing cycle is completed, returns to its starting point, wherein local code makes a call to remote code which returns to the local code upon completion.

3.(previously presented) The method according to claim 1, wherein, during the executing stage the executable application software implements a library for managing a sequence of events corresponding to the above-mentioned at least one diagram, whereby said library constitutes an automaton that manages the sequence of events of the processes and executes operations that check them, whereby the sequences of events of the operations are defined in an application referential, by means of the method, by describing actual data flows.

4.(previously presented) The method according to claim 1, wherein, during the compiling stage or the executing stage, the method employs an engine that includes an executive that is responsible for recognizing a hardware and communication configuration.

5. (currently amended) The method according to claim 4, wherein the engine manages one or more databases according to a data file descriptor that is provided by an application referential comprising a list of information contained in each file and a list of access indices, wherein each of these indices is a list of fields, and links between multiple encodings of a single item in multiple services, multiple sites, or multiple companies.

6. (previously presented) The method according to Claim 5, wherein the databases are synchronized according to a schedule that is determined by the diagram, upon demand, or before certain predefined events.

7. (previously presented) The method according to claim 1, wherein the transcribing each object stage includes, for programming each action:

i) a naming stage during which a name is given to said action;

ii) a function defining stage, during which said action is identified with a task; and

iii) an information defining stage during which the information that will be processed in the action is identified.

8. (previously presented) The method according to Claim 7, wherein the compiling stage replaces the action name that is given by the transcriber during the naming stage with an index in a task table.

9. (previously presented) The method according to claim 1, wherein, during the representing stage and transcribing stage, said at least one diagram corresponds to at least one tree structure in which the nodes and leaves, where the code is implemented, are made up of actions, whereby return values of these actions determine movement in the tree structure.